Improving Uncoordinated Collaboration in Partially Observable Domains with Imperfect Simultaneous Action Communication

Improving Uncoordinated Collaboration in Partially Observable Domains with Imperfect Simultaneous Action Communication

Aris Valtazanos
School of Informatics
University of Edinburgh
Edinburgh, EH8 9AB, UK
a.valtazanos@ed.ac.uk

Mark Steedman
School of Informatics
University of Edinburgh
Edinburgh, EH8 9AB, UK
steedman@inf.ed.ac.uk

Abstract

Decentralised planning in partially observable multi-agent domains is limited by the interacting agents' incomplete knowledge of their peers, which impacts their ability to work jointly towards a common goal. In this context, communication is often used as a means of observation exchange, which helps each agent in reducing uncertainty and acquiring a more centralised view of the world. However, despite these merits, planning with communicated observations is highly sensitive to communication channel noise and synchronisation issues, e.g. message losses, delays, and corruptions. In this paper, we propose an alternative approach to partially observable uncoordinated collaboration, where agents simultaneously execute and communicate their *actions* to their teammates. Our method extends a state-of-the-art Monte-Carlo planner for use in multi-agent systems, where communicated actions are incorporated directly in the sampling and learning process. We evaluate our approach in a benchmark multi-agent domain, and a more complex multi-robot problem with a larger action space. The experimental results demonstrate that our approach can lead to robust collaboration under challenging communication constraints and high noise levels, even in the presence of teammates who do not use any communication.

Introduction

Collaborative planning is an important challenge for many interactive systems, where multiple agents must work together to achieve a common goal. This problem becomes harder when agents do not have the benefit of centralised coordination, or when the task involves collaboration with a priori unknown teammates. For example, consider a rescue scenario where various robots programmed by different engineers are deployed to a disaster site in an emergency situation. In this setting, generating a commonly agreed plan of actions may be infeasible due to tight time constraints and limited knowledge of the environment. Instead, the robots may be forced to plan from an egocentric perspective, by using their own internal models to select robust actions.

When the above constraints on collaboration arise, agents must reason about the actions of their peers by gathering and processing data on their behaviour. In a general multi-agent planning setting with no centralised coordination, there are two types of input that can be acquired by an agent:

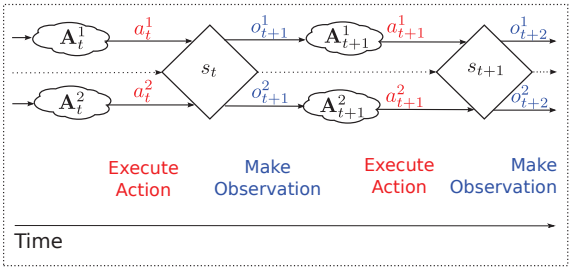
1. Direct **observations** of the teammates, e.g. images from a camera, sonar readings, or other sensory inputs.
2. Inter-agent **communication**, i.e. messages received from teammates about their own (past or future) actions, observations, plans, or intentions.

Each of these input types is significant for uncoordinated collaboration, but also carries its own challenges and limitations. On the one hand, sensory observations are collected and processed internally by each agent, so they are not generated by unknown external protocols or mechanisms. However, many domains of practical interest are characterised by partial and/or limited observability, so agents may not be able to view their teammates (reliably and noiselessly) at all times. On the other hand, a communicated message provides direct insight on the planning process used by the sending agent, thus helping the receiving agent in selecting its own actions with regard to the overall team goal. However, limited bandwidth or poor synchronisation issues may lead to dropped or delayed messages during an interaction. Furthermore, communication channels may be noisy or unreliable, giving rise to misinterpreted (or uninterpreted) messages that also impact an agent's knowledge of its peers.

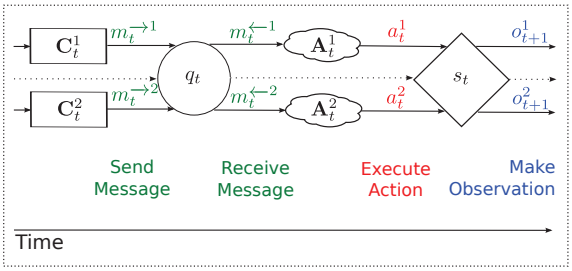
In light of the above constraints, an important challenge in uncoordinated collaboration under partial observability lies in combining the relative merits of observation- and communication-based reasoning. Planning under limited observations has been widely studied using the Partially Observable Markov Decision Process (POMDP) formulation (Kaelbling, Littman, and Cassandra 1998), which however does not explicitly model communication (Figure 1(a)). By contrast, communication-based planning in multi-agent systems is a more open-ended problem, which is typically concerned with the following issues:

1. **When** and **how** to communicate.
2. **What** to communicate.

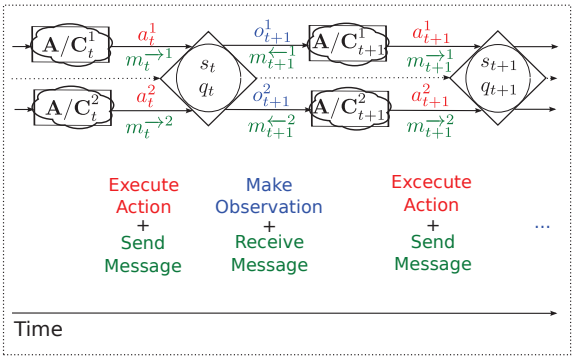
With regard to the first issue, there is a distinction between implementations assuming *perfect synchronisation* between communication and planning phases (where agents can reliably exchange messages before selecting their actions, as in Figure 1(b)), and those accounting for *stochastic communication* (where messages can be lost or delayed). With regard to *what* to communicate, a commonly employed approach is *observation-based* communication (as in the work of Pynadath and Tambe (2002)), where agents exchange their *most recent* observations. The motivation behind this choice is



(a) Planning with no communication (as in standard decentralised POMDP planning implementations).



(b) Planning with distinct (synchronised) action selection and communication phases.



(c) Planning with simultaneous action selection and communication phases (the approach followed in this paper).

Figure 1: Sketch drawings of different approaches to combining decentralised planning and communication in partially observable multi-agent domains. Illustrations are given for the two-agent case, with superscripts ^{1} and ^{2} being the agent indices, and subscripts denoting time. *Clouds:* Action selection/planning steps ($\mathbf{A}$). *Rectangles:* Communication/message selection steps ($\mathbf{C}$). *Diamonds:* State updates. *Circles:* Communication updates. Other notation: actions – a, observations – o, world states – s, message queues – q, sent messages – $m^{\rightarrow}$, received messages – $m_t^{\leftarrow}$.

that agents obtain an approximate world model by combining the locally transmitted views, thus effectively reducing collaboration to a centralised planning problem.

Despite these advantages, we also note some important limitations of observation-based communication. First, planning becomes sensitive to stochastic communication, as delayed or dropped observation messages inevitably lead to an incomplete or outdated world model. Second, even when communication is perfectly synchronised, there is an underlying assumption that all agents use the same planning mechanism (and thus interpret each other's observations identically), which however breaks down when heterogeneous teammates are called to collaborate (as in the rescue scenario example introduced earlier). Third, reasoning about other agents' observations effectively means modeling their own *beliefs* and uncertainty about the world state, which increases the depth of reasoning and thus also the complexity of the planning process.

In this paper, we propose a novel *action-based communication* model for uncoordinated collaboration in partially observable domains. Our approach extends a state-of-the-art online POMDP Monte-Carlo planner with a simple communication protocol, where agents execute and broadcast their selected actions *simultaneously* (Figure 1(c)). Agents maintain a distribution (defined in terms of their *own* beliefs) over selected teammate actions, which is updated when a new message is received. The planner then uses this distribution as a prior in action sampling during Monte-Carlo iterations, and to perform a new type of *factored* policy learning, which decouples observation- and message-based value updates.

As illustrated in Figure 1(c), our protocol implies that transmitted messages are only received *after* the current planning cycle. Thus, even when the communication channel is perfect and noiseless, agents will *always* have delayed information on their peers. This motivates a looser coupling between communication and planning, which, as we demonstrate in our results, makes our approach more robust to three types of noise:

1. Message **losses**.
2. Message **delays**.
3. Message **corruptions/misinterpretations**.

The latter type of noise has received less attention in partially observable multi-agent planning, but we argue that it is particularly important when considering collaboration with heterogeneous agents, such as humans or human-controlled robots. These settings typically involve complex speech generation and recognition processes that significantly constrain communication within a team.

Another distinguishing feature of our approach is that agents do not exchange their observations, and thus also do not explicitly model each other's beliefs and planning mechanisms. This keeps the computational complexity of our approach low and scalable to challenging domains.

In the remainder of this paper, we first review related ideas and techniques from the literature, and we subsequently present our methodology, describing our planning algorithms and communication protocol. We then evaluate our approach in two multi-agent domains; a benchmark box-pushing problem with a small action space, and a more challenging multi-robot kitchen planning scenario. Our results demonstrate that planning with action communication outperforms non-communicative implementations under most noise configurations, while requiring comparable computation time. We conclude by summarising our key contributions and suggesting possible future directions.

Related Work

Single-agent planning

Planning in partially observable single-agent domains is usually described in terms of a Partially Observable Markov Decision Process (POMDP) (Kaelbling, Littman, and Cassandra 1998), $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{Z}, \mathcal{R} \rangle$, where $\mathcal{S}, \mathcal{A}, \mathcal{O}$ are the state, action, and observation sets, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ is the transition function, $\mathcal{Z} : \mathcal{S} \times \mathcal{O} \times \mathcal{A} \mapsto [0, 1]$ is the observation function, and $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \Re$ is the reward function giving the expected payoff for executing an action.

POMDPs can be used to model a wide range of decision problems. However, analytical solutions are known to be hard to compute (Papadimitriou and Tsitsiklis 1987), with several problems requiring hours or even days to solve exactly. This is a restricting factor for systems with tight computational constraints and varying task specifications.

The complexity of finding offline analytical solutions has led to the development of *online* POMDP planning methods, which only consider the current state of the interaction and use limited computation time. Partially Observable Monte Carlo Planning (POMCP) (Silver and Veness 2010) employs Monte-Carlo Tree Search to *sample* the problem space efficiently. This method models action selection as a *multi-armed bandit* problem, by initially estimating the value of random action sequences (referred to as *rollouts*), and then balancing exploration and exploitation through the Upper Confidence Bound (UCB) heuristic (Kocsis and Szepesvári 2006). POMCP has been successfully applied to problems with large branching factors (Gelly et al. 2012), and implemented in a winning entry of the 2011 International Planning Competition (Coles et al. 2012).

Multi-agent planning without communication

A Decentralised POMDP (Dec-POMDP) (Bernstein et al. 2002) is a generalisation of a POMDP to multi-agent systems, defined as $\langle \mathcal{I}, \mathcal{S}, \vec{\mathcal{A}}, \vec{\mathcal{O}}, \mathcal{T}, \mathcal{Z}, \mathcal{R} \rangle$, where $\mathcal{I} = \{1...n\}$ is the set of agents, $\vec{\mathcal{A}} = \times_{i \in \mathcal{I}} \mathcal{A}^i$ is the set of joint actions $\vec{a} = \langle a_1, ..., a_n \rangle$, defined as the Cartesian product of the agents' individual action sets $\mathcal{A}^i$, $\vec{\mathcal{O}} = \times_{i \in \mathcal{I}} \mathcal{O}^i$ is similarly the set of joint observations, with $\mathcal{T}$, $\mathcal{Z}$, and $\mathcal{R}$ defined as in POMDPs, with $\vec{\mathcal{A}}$ and $\vec{\mathcal{O}}$ substituting $\mathcal{A}$ and $\mathcal{O}$.

Compared to POMDPs, Dec-POMDPs carry the additional limitation that action and observation spaces grow exponentially with the number of agents, thus also being intractable. Furthermore, fast single-agent methods like POMCP cannot be directly extended to Dec-POMDPs, due to the added constraint of reasoning about joint observations and beliefs. In this paper, we describe an alternative, egocentric method of adapting POMCP to multi-agent system constraints. Each agent keeps track of and updates values over only its own history and observation space, with teammate actions modeled at the rollout sampling level. This keeps the complexity of the planning process low and scalable to larger and more complex planning spaces.

Multi-agent planning with communication

In their general form, the POMDP and Dec-POMDP formulations do not explicitly model communication between agents. To address this issue, several extensions combining decentralised planning and message passing have been proposed. One of the earlier such approaches is the Communicative Multi-agent Team Decision Problem (Pynadath and Tambe 2002), which presents a general framework for teamwork with instantaneous communication. However, this model assumes distinct pre-communication and post-communication phases (similarly to Figure 1(b)) and perfect noiseless channels without delays and losses.

Becker, Lesser, and Zilberstein (2005) consider communication with associated costs in a Decentralised MDP framework, where agents must additionally decide when to transmit their local *states* to their peers. This concept is extended to partially observable domains by Roth, Simmons, and Veloso (2005), leading to reasoning over joint beliefs based on intermittently transmitted local observations. Despite factoring communication costs, both of these works also assume reliable communication channels, through which agents are able to merge their local observations (or states) and construct a more complete approximation of the world.

Planning with communication costs has also been studied in the context of *coordinated* multi-agent reinforcement learning (Zhang and Lesser 2013). This method uses a *loss rate threshold* to select sub-groups of agents that will coordinate their actions (and communicate) at each time step. Despite addressing concerns of systems with larger numbers of agents, this work makes stronger assumptions on inter-agent coordination, while also not considering noise (and actual message losses) in the communication channel.

The problem of decentralised planning with *imperfect* communication has recently received more attention in the literature. Within the Dec-POMDP framework, Bayesian game techniques (Oliehoek, Spaan, and Vlassis 2007) and tree-based solutions (Oliehoek and Spaan 2012) have been proposed to deal with *one-step message delays*. This is extended to account for stochastic delays that can be longer than one time step (Spaan, Oliehoek, and Vlassis 2008). Our simultaneous communication model (Figure 1(c)) aims to address similar effects, but does not assume any explicit bounds on message delays. Furthermore, we also consider other types of communication noise such as message losses (which are effectively analogous to infinite-time delays).

Wu, Zilberstein, and Chen (2011b) introduce a model of bounded message-passing, where the communication channel may be periodically unavailable. In this context, two distinct protocols are evaluated; the first postpones communication until the channel becomes available again, and the second drops the communication attempt entirely. While these constraints are similar to the ones we consider, we also note some important differences. First, the bounded communication model uses separate communication and action phases, whereas we adopt a more constrained simultaneous approach (Figure 1). Second, the above protocols assume that agents know *when* the communication channel is unavailable; by contrast, our method makes no assumptions on when, if, or how transmitted messages will reach other teammates.

A common feature of all the above works is that agents communicate their local *observations* to each other, with

the goal of combining them and forming a more complete world model. As discussed in the introduction of this paper, we adopt a different, *action*-based communication protocol, which does *not* aim to "centralise" a decentralised decision problem through observation exchange and joint-belief reasoning. Instead, agents maintain their own incomplete views of the world, and use (any) communicated actions received from their teammates to bias their own, egocentric planning process. As we demonstrate in our results, this protocol maintains a robust performance even under high levels of communication channel noise. Moreover, our approach is also tolerant to novel types of noise, such as message corruption, which have so far received little attention in decentralised planning under partial observability.

Collaboration without prior coordination

Another common underlying aspect of the works presented in the previous section is collaboration between *identical* agents. However, many of these approaches break down when the domains feature heterogeneous agents with different planning, reward, or world modeling processes. To address this issue, our method draws inspiration from the *ad-hoc* teamwork problem (Stone et al. 2010), which considers collaboration without pre-coordination in the presence of unknown teammates. In this context, the POMCP algorithm has been combined with Markov games (Wu, Zilberstein, and Chen 2011a) and transfer learning (Barrett et al. 2013b) to generate team-level strategies. However, both of these works assume full world observability and do not involve inter-agent communication.

A communication protocol for ad-hoc teamwork has been proposed by Barrett et al. (2013a), where message selection is integrated within the planning process. In particular, each agent has a fixed set of communicative messages that are synthesised through the POMCP multi-armed bandit framework. Despite taking some important first steps towards combining planning and communication with heterogeneous agents, this work assumes full world observability and noiseless channels, while also using distinct communication and action phases. To our knowledge, our method is the first to address the combined existence of several of the challenges described so far, i.e. uncoordinated collaboration with unknown teammates in partially observable domains, in the presence of imperfect communication.

Method

In this section, we first provide an overview of POMDP and Monte-Carlo planning, summarising some key concepts from Silver and Veness (2010). Then, we extend the single-agent POMCP definitions to model egocentric planning in multi-agent systems. We subsequently present our communication protocol, and then describe our approach to planning with communicated actions. We conclude by providing detailed algorithms for our implementation.

Planning in single-agent POMDPs

Preliminaries An agent acting in a partially observable domain cannot directly observe the state of the world, s_t, but only knows a history of past actions and observations up to the current time t, $h_t = \{o_0, a_0, \ldots, o_t, a_t, o_{t+1}\}$, and plans with respect to the belief $B(s, h)$, which is a history-dependent distribution over states. A policy $\pi(h, a)$ is a mapping from histories to actions, and the *return* $R_t = \sum_{k=t}^{\infty} \gamma^{k-t} r_k$ is the obtained reward starting at time t, where $0 < \gamma \leqslant 1$ is a discount factor, and each r_k is drawn from the reward function $\mathcal{R}$. The *value function* $V^{\pi}(h) = \mathbb{E}[R_t|h]$ is the expected return under π starting at history h, and $V^*(h) = \max_{\pi} V^{\pi}(h)$ is the optimal value function. Additionally, $Q^{\pi}(h, a)$ is the value of taking action a after history h, and then following policy π.

Monte-Carlo planning Due to the complexity associated with computing V^* exactly, POMCP approximates this value through sampling-based forward search from the current history h. The planner uses a black-box simulator $(s_{t+1}, o_{t+1}, r_{t+1}) \sim \mathcal{G}(s_t, a_t)$ that generates successor values given the current state and action. The value of a state s is approximated by the mean return of n *simulations*, or *plan samples*, $V(s) = \frac{1}{n} \sum_{i=1}^{n} R_i$, each searching the problem space over a fixed time horizon $\mathcal{H}$. Starting with 0 values, the planner also maintains visitation counts $N(h), N(h, a)$ and Q-value estimates $Q(h, a)$ for each history-action pair, which are updated during plan sampling; visitation counts are incremented by 1 each time a history or history-action pair is sampled, and Q-values are updated as $Q(h, a) \leftarrow Q(h, a) + \frac{R - Q(h,a)}{N(h,a)}$, where R is the return of the most recent plan sample. When a history h has not been visited before, actions are chosen randomly based on a *rollout* policy, $a \sim \pi_{rollout}(h)$. Otherwise, the optimal action is selected as

$$a^* = \arg\max_{a \in \mathcal{A}} Q(h, a) + c\sqrt{\log(N(h))/N(h, a)}, \quad (1)$$

using the UCB heuristic with an exploration constant c.

Egocentric POMCP for multi-agent systems

Extending POMCP heuristics to multi-agent systems is not straightforward due to the existence of joint actions and observations. For fully observable systems, Eq. 1 can be rewritten as $\vec{a}^* = \arg\max_{\vec{a} \in \vec{\mathcal{A}}} Q(s, \vec{a}) + c\sqrt{\log(N(s))/N(s, \vec{a})}$ (Wu, Zilberstein, and Chen 2011a), replacing histories with states and single-agent actions with joint ones. Unfortunately, this does not apply to partially observable domains with no communication because agents cannot observe joint histories $\vec{h}$ (and actions $\vec{a}$).

To overcome this problem (and avoid maintaining expensive beliefs over the beliefs of others), we restrict our sample updates to single-agent N and Q values as in the original POMCP framework. However, we modify the rollout policy to generate random joint actions, $\vec{a} \sim \pi_{rollout}(h)$, though it is still parametrised only by the planning agent's history h. Similarly, we parametrise the black-box simulator in terms of joint actions, $(s_{t+1}, o_{t+1}, r_{t+1}) \sim \mathcal{G}(s_t, \vec{a}_t)$, though it still generates observations and rewards for the planning agent only. These modifications can be implemented at minimal additional computational cost, while also not making any assumptions about other agents' beliefs and histories.

Simultaneous Action Communication

Communication protocol and structures Despite providing support for decision-making in the presence of other agents, the above definitions do not model communication within a team. To address this issue, we define a simple protocol through which agents can communicate with each other. In particular, let $\mathcal{M}$ denote the set of messages that can be exchanged between agents, $m^{\rightarrow}$ a message *sent* by an agent to its teammates, and $m^{\leftarrow}$ a *received* message. We assume a simple broadcasting protocol, where the world state, s, is augmented with a *message queue*, q, containing all the currently available messages. When an agent sends a message $m^{\rightarrow}$, it simply adds it to the front of q; when an agent receives a message $m^{\leftarrow}$, it marks it for removal and $m^{\leftarrow}$ is erased from q at the end of the current time step.

Message selection, exchange, and interpretation As discussed in previous sections, one distinguishing feature of our approach is that agents communicate their actions (and not their observations), and they do so *simultaneously* with action execution (as illustrated in Figure 1(c)). In this context, an agent selects the action a_t to be executed at time t, an deterministically sets its upcoming message to $m_t^{\rightarrow} = a_t$. Thus, the message set is identical to the action set, i.e. $\mathcal{M} = \mathcal{A}$. Furthermore, we can straightforwardly extend the action rollout policy, $\vec{a} \sim \pi_{rollout}(h)$, to obtain the *joint message rollout* policy, $\vec{m}^{\rightarrow} \sim \mu_{rollout}(\vec{a}) = \vec{a}$.

Similarly to actions, agents receive messages from their teammates simultaneously to making observations on the state of the world. At every observation/message reception phase, each agent receives a set, $\{m^{\leftarrow}\}$, of up to $n-1$ messages, where n is the size of the team (so at most one message per teammate is received). However, the size of $\{m^{\leftarrow}\}$ may be potentially smaller when messages are delayed or dropped. The received messages are interpreted under the assumption that all agents are communicating their actions; we use a simple procedure $a \leftarrow \texttt{ParseAction}(m)$ that converts a message m to an action a. When the channel is reliable, `ParseAction` will return the (correct) action that was originally transmitted by the sending agent. Nevertheless, as discussed in the following section, we also consider the case where messages are corrupted during transmission and thus interpreted incorrectly at the receiving end.

Putting everything together, the black-box $\mathcal{G}$ with simultaneous communication and state updates is rewritten as

$$(s_{t+1}, q_{t+1}, o_{t+1}, r_{t+1}, \{m_{t+1}^{\leftarrow}\}) \sim \mathcal{G}(s_t, q_t, \vec{a}_t, \vec{m}_t^{\rightarrow}) \quad (2)$$

Modeling imperfect communication Our protocol can be extended to account for different types of imperfect communication. When modeling message losses, a transmitted message $m_t^{\rightarrow}$ is dropped with probability $0 \leqslant p(loss) \leqslant 1$, in which case the queue q remains unchanged. For message delays, $m_t^{\rightarrow}$ is added with probability $p(delay)$ to q *after* the other updates for step t are completed, which means that it cannot be used by its teammates at decision step $t+1$. Thus, our notion of delay is ***different*** to definitions assuming distinct action and communication phases. In our framework, ***all*** messages by default arrive with a one (planning) step delay, so our definition of delay refers to an ***additional*** communication lag (leading to an overall delay of at least two planning steps). Finally, a transmitted message is corrupted with probability $p(corrupt)$, in which case the receiving agent interprets it as an action other than the one that was originally sent. For the latter type of noise, the number of possible misinterpretations grows with the action set size.

Planning with Communicated Actions

One important open question in our framework is how to use communicated messages to improve the quality of selected actions. To address this issue, we propose two extensions to the original egocentric POMCP framework. First, we introduce a distribution over communicated messages, and use it as a bias in the teammate action sampling process. Second, we define and learn Q-values over the *message* space, thus obtaining a factored approach to action selection.

Teammate action sampling In the non-communicate egocentric POMCP variant, teammate actions are always sampled based on the random rollout policy $\pi_{rollout}$. However, when communication is available, the received messages can provide better insight on the actions chosen by the other agents. To exploit this feature, we introduce a distribution $A'(h,a)$ over communicated teammate actions for every (single-agent) history h and action a. We model $A'(h,a)$ as an unweighted particle filter that is progressively populated from the received messages (similarly to how the belief distribution $B(h)$ is updated from the generated state samples). When $A'(h,a)$ is non-empty, teammate actions are sampled directly from this distribution, otherwise we use $\pi_{rollout}$ as in the non-communicative approach. Thus, action selection is biased towards the information extracted from the received teammate messages, and the rollout policy serves as a fall-back when communication is limited.

Factored value learning and action selection We model communicated messages a special type of observation, over which a separate set of Q-values is learned and used in action selection. In particular, we define a value $Q(h,a,m)$ (and an associated visitation count $N(h,a,m)$) for every history h, action a, and message m, thus introducing an additional layer in the policy learning hierarchy. Like regular Q-values, message values are updated based on the return R generated by each plan sample, i.e. $Q(h,a,m) \leftarrow Q(h,a,m) + \frac{R - Q(h,a,m)}{N(h,a,m)}$. Moreover, Eq. 1 is updated as

$$a^* = \arg\max_{a \in \mathcal{A}} \big[Q(h,a) + \max_{m \in \mathcal{M}} Q(h,a,m) + c\sqrt{\log(N(h))/N(h,a)} \big] \quad (3)$$

to incorporate the learned message values. This leads to a factored learning and action selection procedure, where the planning agent performs distinct learning updates for the different types of input acquired during the interaction.

Summary of Algorithms

We conclude this section by providing implementations for all the procedures described so far. Algorithm 1 summarises the high-level search algorithm; when a history h has not been visited before, states are sampled from the initial state

Algorithm 1: `SearchWithCommunication`(h)

for $i \leftarrow 1$ **to** *NumPlanSamples* **do**
  if $h = empty$ **then** $s \sim \mathcal{I}_S,\ q \leftarrow \varnothing$
  else $\langle s, q \rangle \sim B(h)$
  `Simulate`$(s, q, h, 0)$
return $\arg\max_{a \in A} \left[Q(h,a) + \max_{m \in \mathcal{M}} Q(h,a,m) \right]$

Algorithm 2: `SelectAction`(h)

return $a^* \leftarrow \arg\max_{a \in \mathcal{A}} \left[Q(h,a) + \max_{m \in \mathcal{M}} Q(h,a,m) + c \cdot \sqrt{\frac{\log(N(h))}{N(h,a)}} \right]$

Algorithm 3: `Rollout`(s, q, h, d)

if $d \geqslant \mathcal{H}$ **then return** 0
$\vec{a} \leftarrow \langle a, a' \rangle \sim \pi_{rollout}(h),\ \vec{m}^{\rightarrow} \sim \mu_{rollout}(\vec{a})$
$\langle \acute{s}, \acute{q}, o, r, \cdot \rangle \sim \mathcal{G}(s, q, \vec{a}, \vec{m}^{\rightarrow})$
return $r + \gamma \cdot$ `Rollout`$(\acute{s}, \acute{q}, hao, d+1)$

Algorithm 4: `Simulate`(s, q, h, d)

if $d \geqslant \mathcal{H}$ **then return** 0
if $N(h) = 0$ **then**
  forall $a \in \mathcal{A}$ **do**
    $N(h,a) \leftarrow 0,\ Q(h,a) \leftarrow 0,\ A'(h,a) \leftarrow \varnothing$
    forall $m \in \mathcal{M}$ **do**
      $N(h,a,m) \leftarrow 0,\ Q(h,a,m) \leftarrow 0$
  return `Rollout`(s, q, h, d)
$a \leftarrow$ `SelectAction`(h)
if $A'(h,a) \neq \varnothing$ **then** $a' \sim A'(h,a)$
else $\langle \cdot, a' \rangle \sim \pi_{rollout}(h)$
$\vec{m}^{\rightarrow} \sim \mu_{rollout}(\langle a, a' \rangle)$
$\langle \acute{s}, \acute{q}, o, r, \{m^{\leftarrow}\} \rangle \sim \mathcal{G}(s, q, \langle a, a' \rangle, \vec{m}^{\rightarrow})$
$R \leftarrow r + \gamma \cdot$ `Simulate`$(\acute{s}, \acute{q}, hao, d+1)$
$B(h) \leftarrow B(h) \cup \langle s, q \rangle,\ N(h) \leftarrow N(h) + 1$
$N(h,a) \leftarrow N(h,a)+1, Q(h,a) \leftarrow Q(h,a) + \frac{R - Q(h,a)}{N(h,a)}$
forall $m \in \{m^{\leftarrow}\}$ **do**
  $N(h,a,m) \leftarrow N(h,a,m) + 1$
  $A'(h,a) \leftarrow A'(h,a) \cup$ `ParseAction`(m)
  $Q(h,a,m) \leftarrow Q(h,a,m) + \frac{R - Q(h,a,m)}{N(h,a,m)}$
return R

distribution $\mathcal{I}_S$, and the message queue is empty. Algorithm 2 recaps the communication-based action selection formulation, and Algorithm 3 gives the random rollout sampling procedure. Finally, Algorithm 4 illustrates the main simulation algorithm, where Q-values are initialised and updated.

Results

We evaluate our approach in two multi-agent domains; a benchmark cooperative box-pushing problem from the Dec-POMDP literature, and a more complex multi-robot kitchen scenario with a significantly larger action space. The domains are noisy, so actions and observations are perturbed with 0.1 probability. In both problems, we compare a decentralised POMCP agent with simultaneous action communication (denoted **SAC**) to an identical agent with no communication (**NoComm**); both agents follow the planning approach defined in the previous section, but **NoComm** does not learn or use any message Q-values.

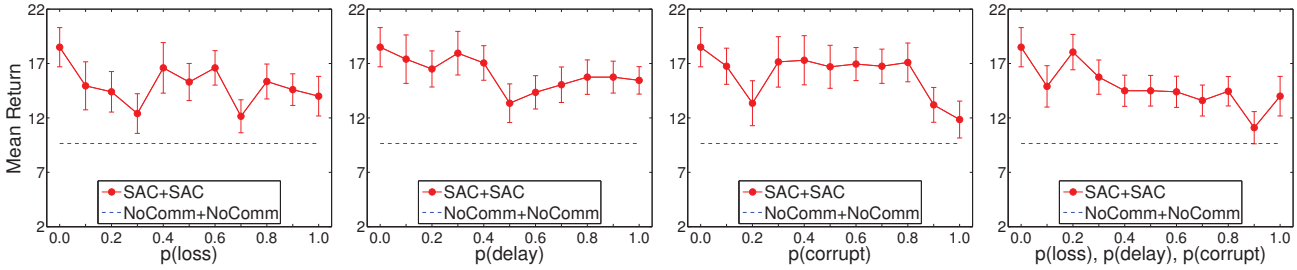
We assess the two algorithms in two-agent teams, dividing our experiments for each domain in two phases. In the first, *same-agent team* phase, we pair **CAC** and **NoComm** with an identical agent, and compare the resulting teams under varying message loss, delay, and corruption probabilities. We test for the cases where each threshold ($p(loss)$, $p(delay)$, $p(corrupt)$) is modified independently, and the case where all types of noise are combined. In the second, *heterogeneous-agent team* phase, we fix the three noise thresholds to 0.1 (so they are equal to the action and observation noise probabilities), and compare performance in collaboration with other, unknown agents. In this context, the candidate teammates are an agent selecting random actions (**Rand**), and a *problem-specific* human-designed agent (**HumDes**) running a robust hand-coded algorithm. Both **Rand** and **HumDes** can communicate their actions to their teammates, though their behaviour does not make any assumptions about the availability of communication.

To demonstrate the generality of our approach, we use the same experiment parameters in both problems. For each team, we average results over 100 runs with 1024 plan samples per decision step, recording the **mean return** (the reward achieved by the team after each run) and the average computation **time** *per team per step*. We set the time horizon to $\mathcal{H} = 20$ and the exploration constant to $c = r_{max}$, where r_{max} is the maximum reward of the domain. Experiments were run on a dual core 3GHz PC with a 4GB RAM.

Cooperative box-pushing

In the cooperative box-pushing domain (Seuken and Zilberstein 2007), agents interact in a walled grid with one large and two small boxes. Agents get a reward of +10 for pushing a small box to the edge of the grid and +100 for doing this for the large box. However, the large box can be moved only if simultaneously pushed by both agents. Each agent has 4 actions (*move*, *turn_left*, *turn_right*, *stay*) and can only see the square to its front, with the possible observations being *empty*, *other_agent*, *wall*, *small_box*, *large_box*. Each agent gets a reward of -0.1 for every step taken, and -5 for bumping into a wall, its teammate, or a box it cannot move. When any box reaches the edge, the problem resets to the start state and the interaction repeats until the time horizon is reached.

The box-pushing results for same-agent teams under different types of communication noise are presented in Figure 2. The **SAC + SAC** team is seen to outperform the **NoComm** variant under all possible noise thresholds, even when the communication channel is always unavailable or unreliable. Moreover, the performance is similar across the different types of noise (and the case where all types of noise combine), thus indicating that our method is not sen-



(a) Message losses (b) Message delays (c) Message corruptions (d) All types of noise combined

Figure 2: Box-pushing domain - comparison of same-agent teams (**SAC + SAC** and **NoComm + NoComm**) for different types of communication channel noise. $p(loss)$: probability of message loss. $p(delay)$: probability of message delay. $p(corrupt)$: probability of message corruption. (a)-(c): Returns obtained under a single type of noise - the other probabilities are set to 0. (d): All types of noise combined (with $p(loss) = p(delay) = p(corrupt)$ in each case).

sitive to any specific irregularities. Thus, the simultaneous action communication approach benefits from the exchange of messages when the channel is reliable, while not being impacted by message losses, delays, or corruptions even under severely restricted communication conditions.

An experimental evaluation of decentralised planning with communication in the box pushing domain has also been conducted by Wu, Zilberstein, and Chen (2011b). In their results, they report considerably higher positive returns for most noise thresholds, which however drop to negative values when the channel is always unavailable (whereas our method still manages to achieve a positive mean return). Nevertheless, a direct comparison with simultaneous action communication is problematic for two reasons. First, as discussed in the related work section, their approach uses distinct action execution and communication phases, where successfully transmitted messages always provide up-to-date information on teammate observations. In our framework, even where there is no additional noise, all messages arrive with a one-step delay. Thus, our experimental setting introduces considerably harder constraints on collaboration that are not fully captured by the distinct phase model[1]. Second, their method first solves a centralised MDP version of the problem, and then uses it as a heuristic in the decentralised algorithms, thus improving planning performance. By contrast, our approach uses no such heuristics, following a fully uncoordinated planning approach that does not incorporate any prior centralised knowledge.

The results for collaboration with heterogeneous agents in the box-pushing domain are given in Figure 3. Compared to **NoComm**, the **SAC** agent achieves better returns when paired with all other teammates. Moreover, the **SAC + NoComm** team outperforms both the **SAC + SAC** and the **NoComm + NoComm** combinations, thus indicating that our method can achieve robust collaboration even with agents who do not use any communication.

[1]In practice, the distinct and simultaneous phase models are aligned only in the maximal 1.0 loss rate case, when all messages are dropped in both frameworks (in this case, only our method attains positive returns in the box-pushing problem). For all other noise levels < 1.0, the distinct-phase framework provides agents with synchronised messages for at least some fraction of the time; this advantage would however be lost in our experimental setting.

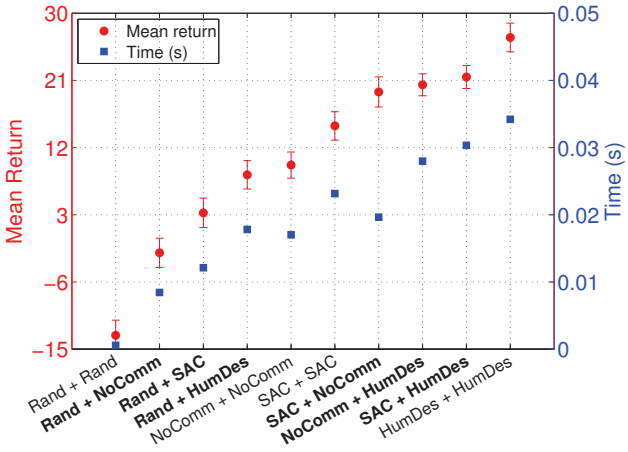
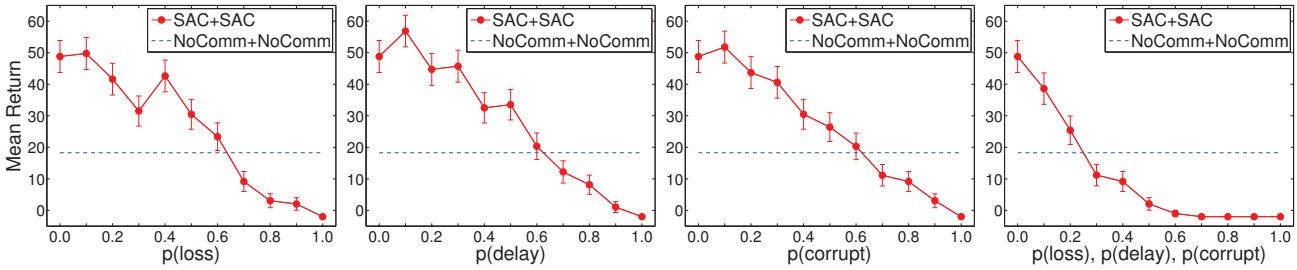


Figure 3: Box pushing domain - results for heterogeneous agent teams. The results are sorted in order of increasing mean return. *Boldface labels:* teams with different agents. *Non-boldface labels:* teams with the same agents. Communication noise: $p(loss) = p(delay) = p(corrupt) = 0.1$. See text for description of the different algorithms.

Multi-robot kitchen domain

The multi-robot kitchen domain is an extension of a single-agent problem described by Petrick, Geib, and Steedman (2009). In the multi-agent variant, two bi-manual robots are tasked with transporting a tray between two different kitchen locations. The kitchen has five locations (*sideboard*, *stove*, *fridge*, *dishwasher*, *cupboard*), and each robot can *move* between them. A location can be *open*ed or *close*d by a robot's left or right hands. The tray can be *grasp*ed or *put_down* at a location, or *transport*ed between locations; these actions are joint and fail if not simultaneously executed by both robots. If a joint action fails at the start location, the tray is dropped and needs to be *placed_upright* by one robot; if it fails at any



(a) Message losses (b) Message delays (c) Message corruptions (d) All types of noise combined

Figure 4: Kitchen domain - results for different types of communication noise. See caption of Figure 2 for further explanations.

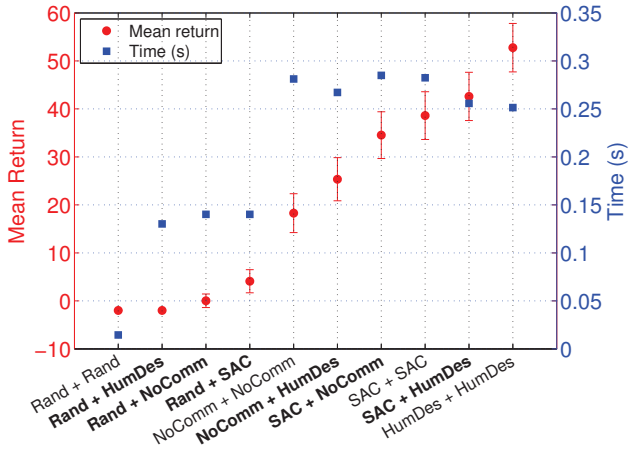


Figure 5: Kitchen domain - results for heterogeneous agent teams. See caption of Figure 3 for further explanations.

other location, the tray is moved back to the start. The tray and teammate are visible only when in the same location as the planning robot. The reward is +100 for successfully taking the tray to the goal, and -0.1 for every other step.

When aggregating all possible object/location/gripper combinations, the kitchen domain has a total of 175 actions per agent. This represents a considerably larger problem space than the box-pushing domain, which impacts both planning and communication (since, as previously discussed, the number of possible message misinterpretations grows with the action space). Additionally, several *distinct* joint actions are now needed to achieve the goal, i.e. *grasp*, *transport*, and *put_down* (as opposed to just *moving* the large box). Another distinguishing feature is that no goal can now be attained by a single agent (as with small boxes previously), so robots *must* collaborate to get a positive reward.

The higher difficulty of the kitchen domain is illustrated in Figure 4, where the **SAC** algorithm now performs worse than **NoComm** in some of the more restricting noise cases. This is particularly evident in Figure 4(d), where the performance decline is more rapid. Nevertheless, even in this challenging problem, **SAC** outperforms **NoComm** under limited communication noise, while exhibiting comparable sensitivity to the different noise types (with 0.6 being the cut-off probability threshold in Figures 4(a), 4(b), and 4(c)).

The **SAC** agent also maintains its ability to achieve superior collaboration with heterogeneous agents than **NoComm** (Figure 5). When comparing with the box-pushing problem results, **SAC** now also outperforms **HumDes** when paired with **Rand** and **NoComm**, thus indicating better adaptation to unknown teammates in this more challenging domain. Furthermore, the **SAC** approach demonstrates comparable efficiency to the other algorithms, as indicated by the recorded computation times.

Conclusions

In this paper, we introduce a novel approach to collaboration in partially observable domains, which is based on the simultaneous execution and exchange of actions between teammates. We extend a state-of-the-art, single-agent Monte-Carlo planner to support egocentric reasoning in multi-agent systems, where communicated messages are used to bias the sampling process and learn policies through factored value updates. Thus, unlike many existing methods that rely heavily on observation and belief synchronisation within a team, our work assumes a looser coupling between planning and communication phases. As demonstrated by our results, our approach outperforms a non-communicative variant in a benchmark domain under varying noise types (message losses, delays, corruptions), while achieving robust collaboration with unknown teammates even in a larger and more complex collaborative planning problem.

We are currently working on integrating communication-based planning with reinforcement learning techniques that actively model the rewards of interacting agents. Our goal is to develop fast, robust decentralised planning algorithms that can be applied to challenging problems with varying task specifications and team compositions. In particular, we are interested in collaborative human-robot interaction applications requiring heterogeneous agents to work (and communicate) in teams towards a common goal, under limited resources and tight coordination constraints.

Acknowledgment

This work has been funded by the European Commission through the EU Cognitive Systems and Robotics project Xperience (FP7-ICT-270273).

References

Barrett, S.; Agmon, N.; Hazon, N.; Kraus, S.; and Stone, P. 2013a. Communicating with Unknown Teammates. In *AAMAS Adaptive Learning Agents (ALA) Workshop*.

Barrett, S.; Stone, P.; Kraus, S.; and Rosenfeld, A. 2013b. Teamwork with Limited Knowledge of Teammates. In *AAAI*.

Becker, R.; Lesser, V.; and Zilberstein, S. 2005. Analyzing Myopic Approaches for Multi-Agent Communication. In *Proceedings of Intelligent Agent Technology*, 550–557.

Bernstein, D. S.; Givan, R.; Immerman, N.; and Zilberstein, S. 2002. The Complexity of Decentralized Control of Markov Decision Processes. *Math. Oper. Res.* 27(4):819–840.

Coles, A. J.; Coles, A.; Olaya, A. G.; Celorrio, S. J.; López, C. L.; Sanner, S.; and Yoon, S. 2012. A Survey of the Seventh International Planning Competition. *AI Magazine* 33(1).

Gelly, S.; Kocsis, L.; Schoenauer, M.; Sebag, M.; Silver, D.; Szepesvári, C.; and Teytaud, O. 2012. The grand challenge of computer Go: Monte Carlo tree search and extensions. *Commun. ACM* 55(3):106–113.

Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence* 101(1-2):99–134.

Kocsis, L., and Szepesvári, C. 2006. Bandit Based Monte-Carlo Planning. In *European Conference on Machine Learning (ECML)*, 282–293.

Oliehoek, F. A., and Spaan, M. T. J. 2012. Tree-Based Solution Methods for Multiagent POMDPs with Delayed Communication. In *AAAI*.

Oliehoek, F. A.; Spaan, M. T. J.; and Vlassis, N. 2007. Dec-POMDPs with delayed communication. In *AAMAS Workshop on Multi-agent Sequential Decision Making in Uncertain Domains*.

Papadimitriou, C., and Tsitsiklis, J. N. 1987. The Complexity of Markov Decision Processes. *Math. Oper. Res.* 12(3):441–450.

Petrick, R.; Geib, C.; and Steedman, M. 2009. Integrating Low-Level Robot/Vision with High-Level Planning and Sensing in PACO-PLUS. *Technical Report, PACO-PLUS Project Deliverable 4.3.5* (available at `http://www.paco-plus.org`).

Pynadath, D. V., and Tambe, M. 2002. The Communicative Multiagent Team Decision Problem: Analyzing Teamwork Theories and Models. *J. Artif. Intell. Res. (JAIR)* 16:389–423.

Roth, M.; Simmons, R.; and Veloso, M. 2005. Reasoning About Joint Beliefs for Execution-time Communication Decisions. In *AAMAS*, 786–793.

Seuken, S., and Zilberstein, S. 2007. Memory-Bounded Dynamic Programming for DEC-POMDPs. In *IJCAI*.

Silver, D., and Veness, J. 2010. Monte-Carlo Planning in Large POMDPs. In *NIPS*, 2164–2172.

Spaan, M. T. J.; Oliehoek, F. A.; and Vlassis, N. A. 2008. Multiagent Planning Under Uncertainty with Stochastic Communication Delays. In *ICAPS*, volume 8, 338–345.

Stone, P.; Kaminka, G. A.; Kraus, S.; and Rosenschein, J. S. 2010. Ad Hoc Autonomous Agent Teams: Collaboration without Pre-Coordination. In *AAAI*.

Wu, F.; Zilberstein, S.; and Chen, X. 2011a. Online Planning for Ad Hoc Autonomous Agent Teams. In *IJCAI*, 439–445.

Wu, F.; Zilberstein, S.; and Chen, X. 2011b. Online Planning for Multi-Agent Systems with Bounded Communication. *Artificial Intelligence* 175(2):487–511.

Zhang, C., and Lesser, V. R. 2013. Coordinating multi-agent reinforcement learning with limited communication. In *AAMAS*, 1101–1108.